UNSOLVABLE PROBLEMS FOR SLDNF RESOLUTION

JOHN C. SHEPHERDSON

▷ Shepherdson showed that there are maximal safe computation rules R_m for SLDNF resolution, such that if a query succeeds under any safe rule it succeeds under R_m, and if it fails under any safe rule it fails under R_m. Later he showed that such maximal rules also get all possible answers, i.e., if a query succeeds with answer θ under any safe rule, then it succeeds with answer equivalent to θ under R_m. The question was raised there, whether there were always *recursive* maximal rules. We answer this negatively. We also show the unsolvability of the problems of deciding whether a query leads to a dead end, whether the Clark completion comp(P) of a program P is consistent, and whether a query which is known to be a logical consequence of the program succeeds in PROLOG. ◁

As in Börger [1], we use the fact [5, 7] that 2-register machines are capable of computing all partial recursive functions, i.e., that given a one argument partial recursive function f, there is a program P using two natural number variables X, Y with p instructions of the form

$X := X + 1$,

If $X \neq 0$ then $X := X - 1$ and goto j,

and similarly for Y, which, started with $X = 2^n$, $Y = 0$, will halt [by reaching the nonexistent $(p+1)$st line] with $X = 2^{f(n)}$ if $f(n)$ is defined, and will not halt if $f(n)$ is undefined [5]. We convert this into a definite Horn clause program with binary predicates $l_1, \ldots, l_{p+1}$, unary predicate s, and constant 0, by replacing a

Address correspondence to John C. Shepherdson, Mathematics Department, University of Bristol, University Walk, Bristol BS8 1TW, England, or Shepherdson@UK.Bristol.qgb.
Received March 1988; accepted August 1988.

655 Avenue of the Americas, New York, NY 10010 0743-1066/91/$3.50

program line

$i: X := X + 1$

with a clause

$$l_i(X,Y) \leftarrow l_{i+1}(s(X),Y),$$

and a program line

i: If $X \neq 0$ then $X := X - 1$ and goto j

with two clauses

$$l_i(s(X),Y) \leftarrow l_j(X,Y),$$

$$l_i(0,Y) \leftarrow l_{i+1}(0,Y),$$

and similarly for instructions involving the variable Y. Finally add a clause

$$l_{p+1}(X,Y) \leftarrow .$$

It is easily seen that the query $\leftarrow l_1(s^{2^n}(0),0)$ succeeds if $f(n)$ is defined, and otherwise has an infinite derivation tree (consisting of a single branch).

Now let A and A' be two recursively inseparable r.e. sets (i.e. [6] such that there is no recursive set B with $A \subseteq B$, $A' \subseteq \bar{B}$) enumerated by g, g' respectively. Take

$$f(x) = \mu y(g(y) = x), \qquad f'(x) = \mu y(g'(y) = x),$$

where $\mu y \Phi(y)$ denotes the least y satisfying $\Phi(y)$. Then $f(x)$ is defined iff $x \in A$, and $f'(x)$ is defined iff $x \in A'$. Take a program P for f as above, and a similar program P' with new predicates l_i' for f'. Add a new unary predicate p and a clause

$$p(X) \leftarrow \neg l_1(X,0), \neg l_1'(X,0),$$

and consider the query

$$Q_n: \leftarrow \neg p(s^{2^n}(0)).$$

By a recursive computation rule we mean an algorithm for selecting a literal in the current goal, given the previous history of the computation. Following Clark [2], we consider only rules R which are *safe* in the sense of Lloyd [4], i.e.

(a) R only selects negative literals which are ground;

(b) having selected a ground negative literal $\neg A$ in some goal, R attempts to finish the construction of a finitely failed SLDNF tree with root $\leftarrow A$ before continuing the remainder of the computation.

We say R is *maximal for success* (*failure*) if a query succeeds (fails) under R when there is any safe rule under which it succeeds (fails).

As far as Q_n is concerned, a safe rule is determined by its choice of first or second literal in the second step, because from then on there is no choice, there being only one literal in the goal under consideration. A recursive safe rule R determines some recursive set B such that R chooses the first literal if n belongs to B, and the second literal otherwise. If you choose the first literal, then if $n \notin A$ the computation is infinite, and if $n \in A$ then Q_n succeeds. Similarly for the second literal. So a rule which is maximal for success must choose the first literal if $n \in A$

and the second if $n \in A'$. So it cannot be recursive, for then the set B would separate A, A'.

NOTES

(1) It is safeness condition (b), which is usually built in to the definition of SLDNF resolution, that is the stumbling block. If that is removed, there are recursive maximal rules. The obvious, but impractical, one is just to start enumerating all possible derivation trees for the main and all subsidiary derivations arising from the use of negation as failure. If there is one which gives success or failure, you will eventually find it. It is possible there might be a feasible procedure where, having used negation as failure on a negative literal, you do not blindly go on and on, but keep popping back to the main derivation and trying the effect of choosing other literals.

(2) For the definite Horn case, using SLD resolution, all rules are maximal for success, and Lassez and Maher [3] have shown that all *fair* rules are maximal for failure as well. And there *are* recursive fair rules, e.g. the one mentioned by Lloyd [4]. This doesn't conflict with the result above, since negative literals do not occur, so that safeness condition (b) is always satisfied.

(3) In our example the goal Q_n succeeds iff $\neg\, p(s^{2^n}(0))$ is a logical consequence of the completion of the program. This is a case where SLDNF resolution is complete but not feasible. However, some variant of SLDNF resolution with safeness condition (b) removed might be feasible, as suggested above. In this example all you need to do is keep switching between the derivation trees for the two literals involved.

(4) The same example, or the simpler one with just P and the query $\neg\, l_1(S^n(0),(0))$, shows the unsolvability of the problem of deciding whether a query dead-ends in a given program with a given computation rule (or with all computation rules, or with some computation rule). Here a dead end is defined as the selection of a ground literal $\neg\, A$ where A neither succeeds nor fails. Of course this is not surprising, being just another aspect of the undecidability of Horn clause logic, i.e. the unsolvability of whether a query succeeds.

(5) Similarly for the unsolvability of the problem of deciding whether comp(P) is consistent. Just take the first program P for an f whose domain of definition is nonrecursive, and add the clause

$$q \leftarrow \neg\, q, l_1(s^{2^n}(0),0).$$

The completion of this program is equivalent to the completion of P together with the statements $\neg\, q$ and $\neg\, l_1(s^{2^n}(0),0)$, so it is consistent iff $f(n)$ is undefined.

(6) Similarly, for the problem of deciding, for a fixed program P and arbitrary goal $\leftarrow A$, where A is known to be a logical consequence of the definite Horn clause program P, whether the goal $\leftarrow A$ succeeds in PROLOG. Just take the program P for an f with nonrecursive domain of definition, and

add a last clause to the program:

$l_1(X,Y) \leftarrow .$

Now $l_1(s^n(0),0)$ is a logical consequence of the program, but it succeeds in PROLOG iff $f(n)$ is defined, because the last line of the program is never reached.

I would like to thank an anonymous referee for many improvements in presentation.

REFERENCES

1. Börger, E., Unsolvable Decision Problems for Prolog Programs, in: G. Goos and J. Hartman (eds.), *Computation Theory and Logic*, Lecture Notes in Comput. Sci. 270, Springer-Verlag, Berlin, 1987.
2. Clark, K. L., Negation as Failure, in: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum, New York, 1987, pp. 293–322.
3. Lassez, J.-L. and Maher, M. J., Closures and Fairness in the Semantics of Programming Logic, *Theoret. Comput. Sci.* 29:167–184 (1984).
4. Lloyd, J. W., *Foundations of Logic Programming*, 1st ed., Springer-Verlag, Berlin, 1984.
5. Minsky, M. L., Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in the Theory of Turing Machines, *Ann. of Math.* 74:437–455 (1961).
6. Rogers, H., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.
7. Shepherdson, J. C. and Sturis, H. E., Computability of Recursive Functions, *J. Assoc. Comput. Mach.* 10:217–255 (1984).
8. Shepherdson, J. C., Negation as Failure: A Comparison of Clark's Completed Data Base and Reiter's Closed World Assumption, *J. Logic Programming* 1(1):51–81 (1984).
9. Shepherdson, J. C., Negation as Failure II, *J. Logic Programming* 3:185–202 (1985).